Розробка бібліотеки підвищення відмовостійкості в мікросервісній архітектурі

•••

Магістерська робота

Виконав: Папроцький Ігор
Керівник магістерської роботи: доктор техн. наук, доцент Глибовець А. М.
Спеціальність: Інженерія програмного забезпечення, МП-2

Постановка задачі

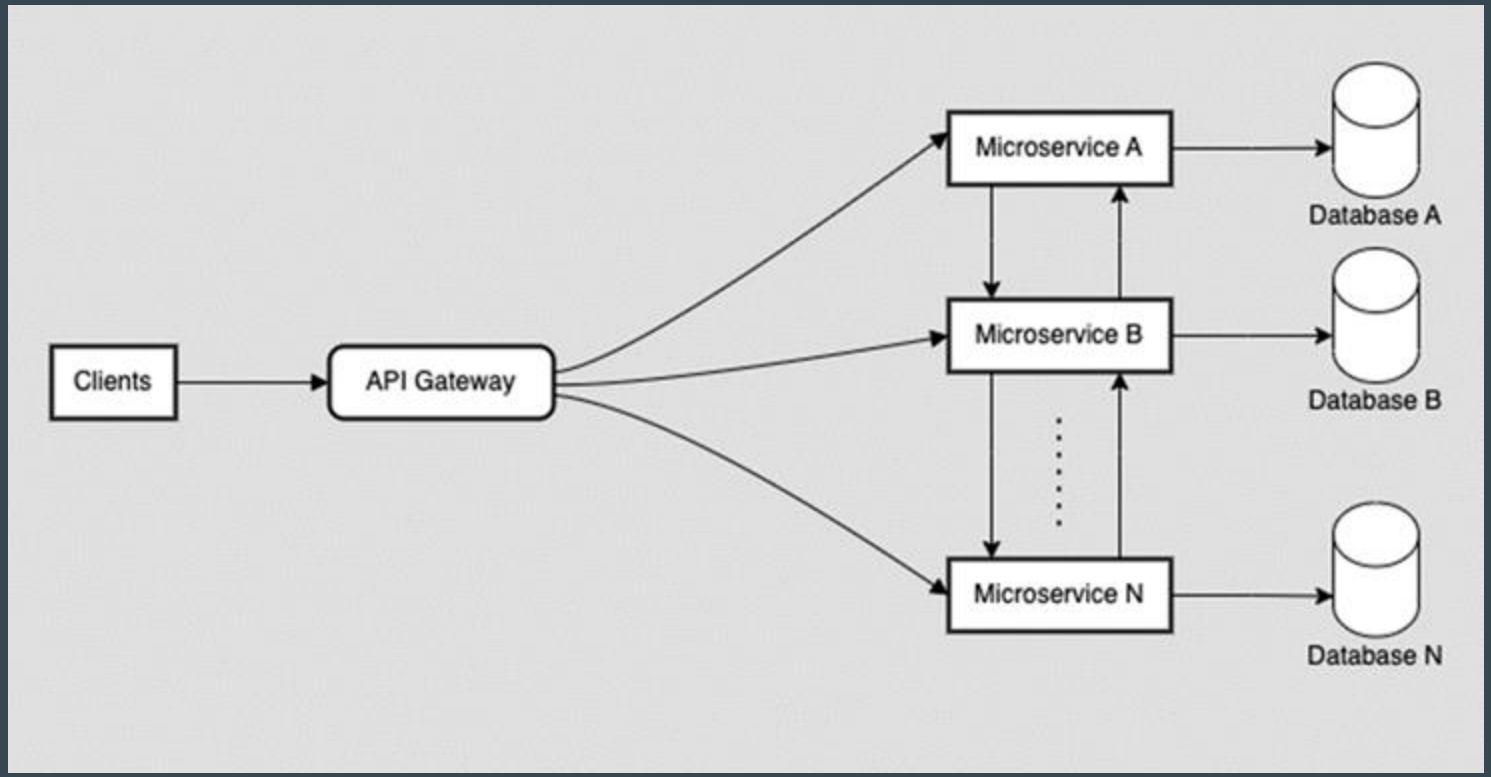
1. Дослідити сучасні підходи із гарантування відмовостійкості під час проектування та розробки програмного забезпечення з використанням мікросервісної архітектури.

2. Провести аналіз сучасних підходи та фреймворків із забезпечення відмовостійкості мікросервісів.

3. Розробити власну реалізацію гарантування відмовостійкості на основі проведеного аналізу.

4. Провести аналіз ефективності розробленої імплементації.

Огляд предметної області: мікросервісна архітектура



Огляд предметної області: відмовостійкість

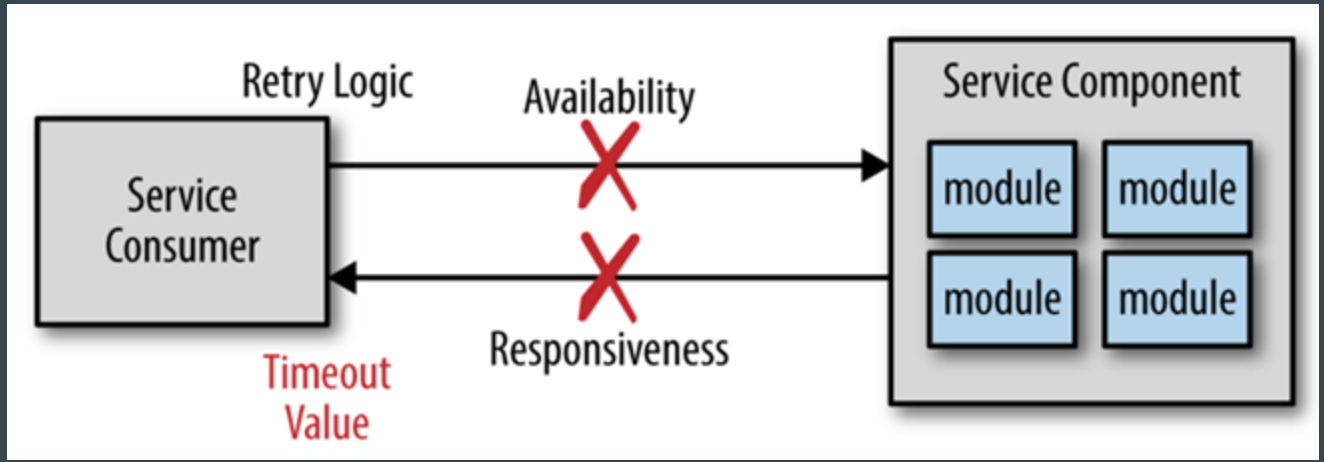
Сфери відмовостійкості в інформаційних технологіях:

- Hardware Fault Tolerance
- Information Redundancy
- Fault-Tolerant Networks
- Fault Detection in Cryptographic Systems
- **Software Fault Tolerance**

Відмовостійкість в програмному забезпеченні

Основні Fault-Tolerance Патерни

- Retry
- Timeouts
- Bulkhead
- Rate Limiter
- Circuit Breaker

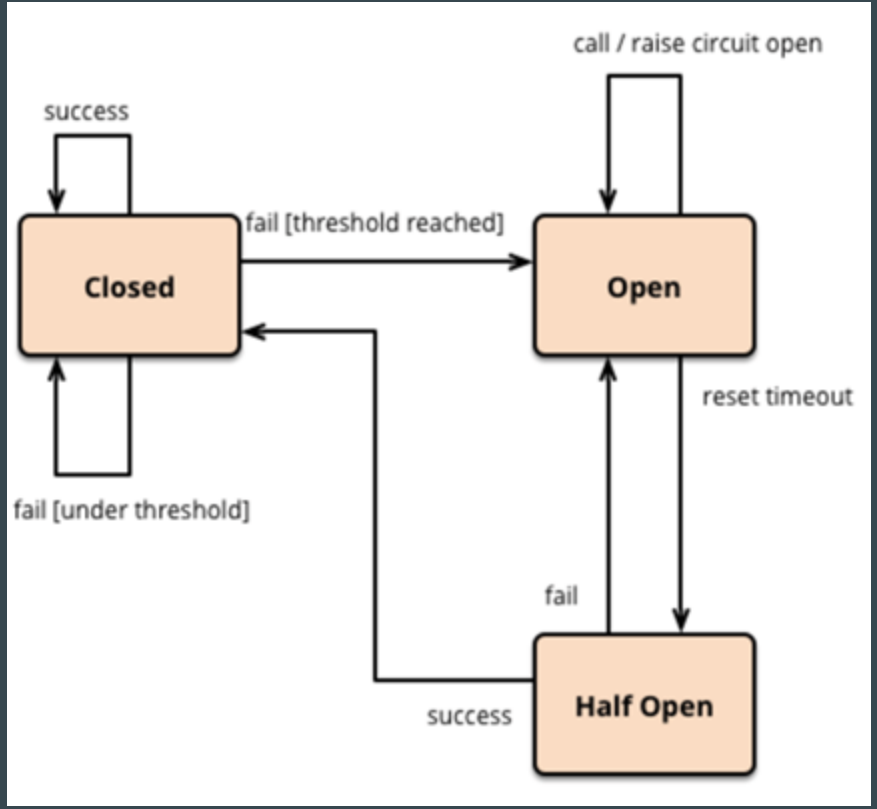


Відмовостійкість в програмному забезпеченні

Патерн Circuit Breaker

Включає в себе:

- 3 стани
- Налаштування порогових значень
- Налаштування затримок (timeout)
- fallback функції
- фільтрація помилок та виключень

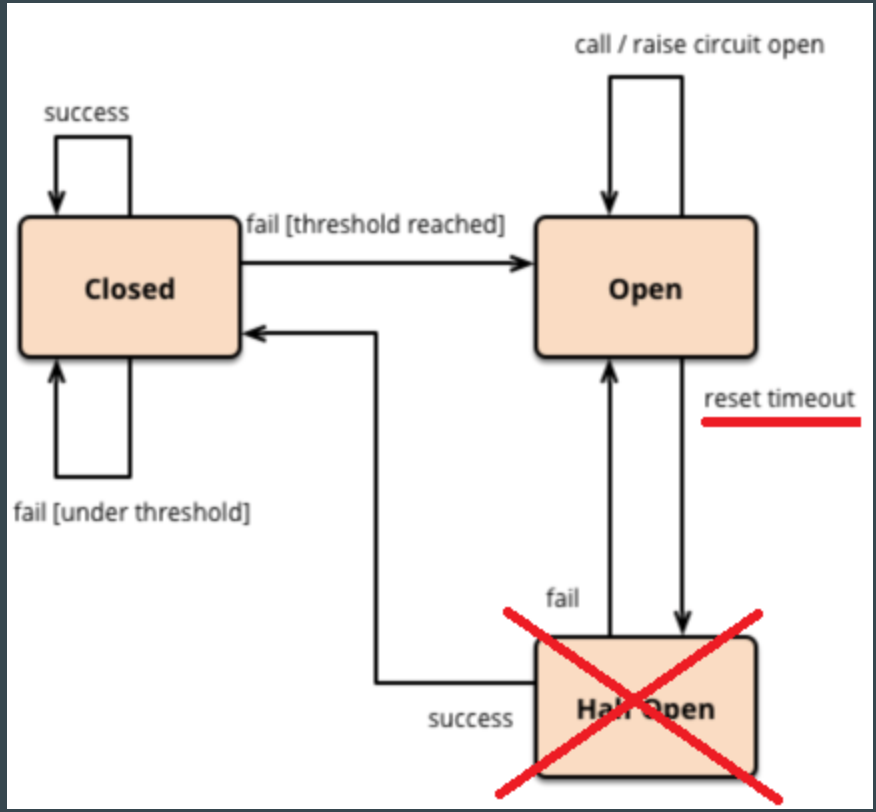


Fault-Tolerance бібліотеки та фреймворки

*Більше не підтримується

Circuit Breaker: модель покращення

- Надлишковість часу очікування у стані Open
- Перехід до моделі у двох станах (Closed <-> Open)
- Прогнозування повернення у стан Closed на основі метричних даних
- Гнучкість підходу



Обчислення прогнозу

$$rating = c1 * (1 - FR) + c2 * (1 - SCR) + c3 * SR + c4 * TOS/MOT$$

$$\text{при } c1 + c2 + c3 + c4 = 1,$$

$$FR, SCR, SR, TOS/MOT \in [0; 1]$$

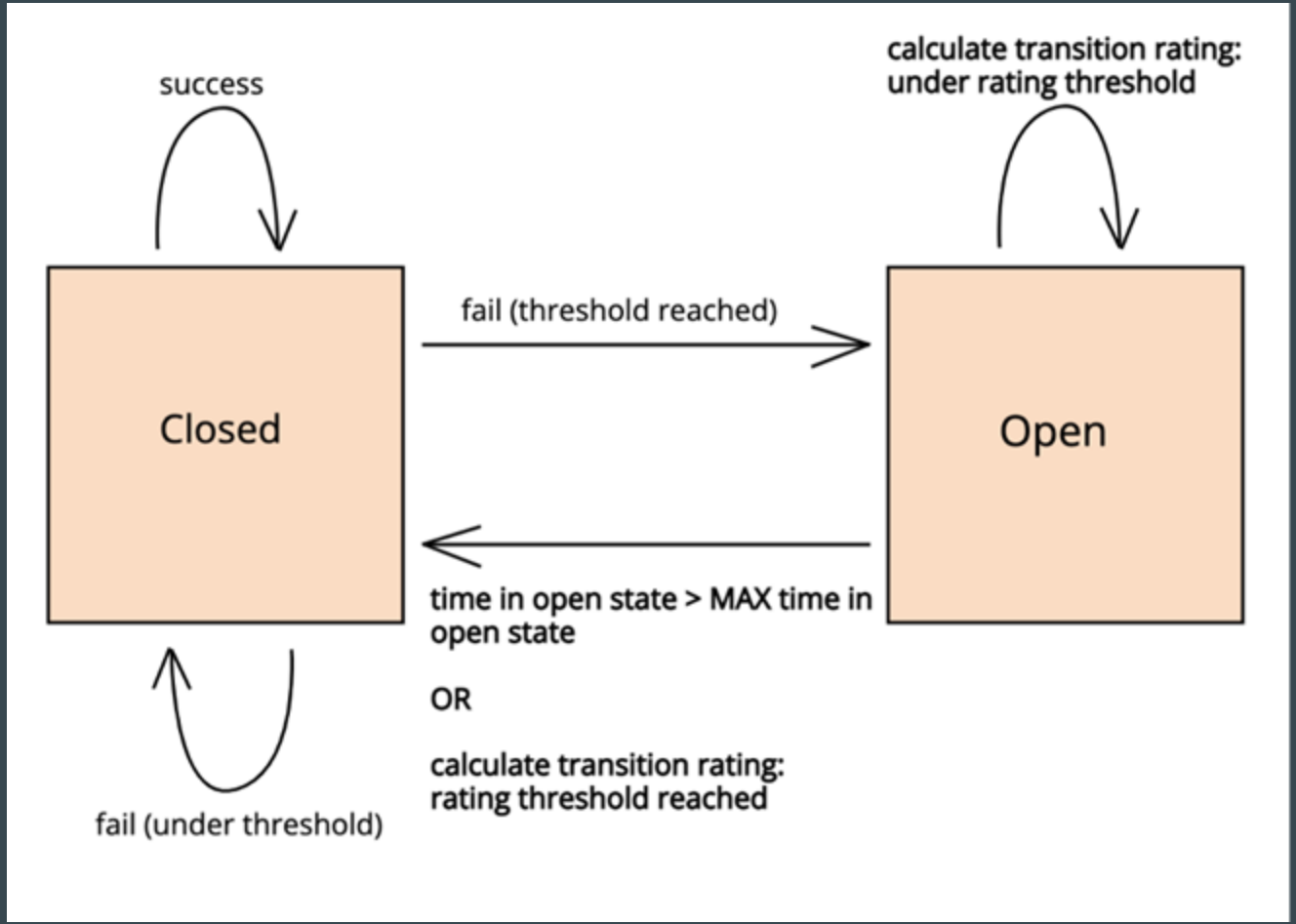
де FR – Failure Rate,
SCR – Slow Call Rate,
SR – Success Rate,
TOS – Time in Open State,
MOT – Maximum Open Time – максимальний час перебування в стані Open, заданий конфігурацією застосунку,
c1, c2, c3, c4 – відповідні коефіцієнти, задаються конфігурацією застосунку

Реалізація

- Мова програмування - Java
- Основа реалізації - розширення бібліотеки Resilience4j
- Легка інтеграція зі Spring Boot
- Підтримка конфігурації (наприклад, порогових значень) фреймворку

Реалізація

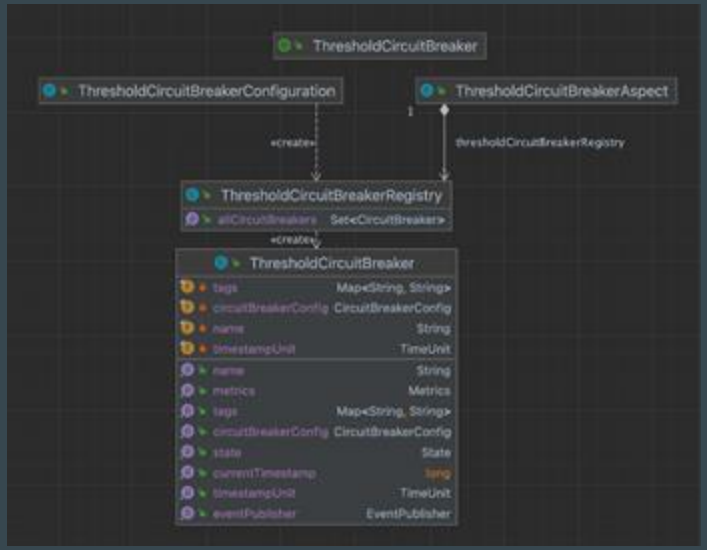
- Схожий життєвий цикл Circuit Breaker, але у 2-х станах

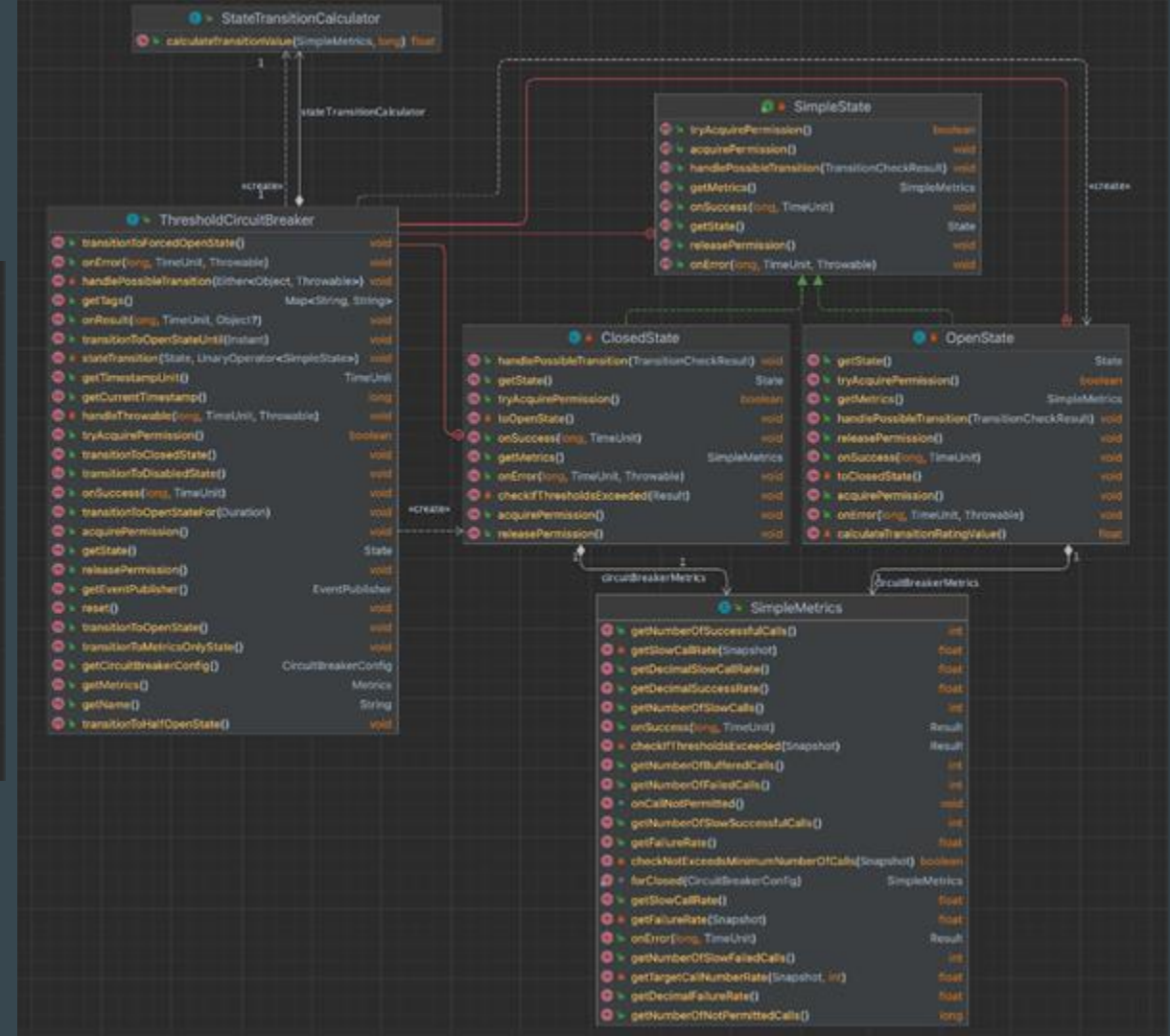


Реалізація

```
public float calculateTransitionValue(@NonNull SimpleMetrics metrics,
                                      long currentOpenStateDurationInNanos) {
  //always close the circuit breaker when the time in open state is longer than the given threshold
  if (currentOpenStateDurationInNanos > DEFAULT_OPEN_STATE_DURATION_THRESHOLD) {
    return Float.POSITIVE_INFINITY;
  }

  float decimalFailureRating = (1 - metrics.getDecimalFailureRate()) * FAILURE_RATE_COEFFICIENT;
  float decimalSlowCallRating = (1 - metrics.getDecimalSlowCallRate()) * SLOW_CALL_RATE_COEFFICIENT;
  float decimalSuccessCallRating = metrics.getDecimalSuccessRate() * SUCCESS_CALL_RATE_COEFFICIENT;

  //time in open state
  float timeInOpenStateRating = (float)
      currentOpenStateDurationInNanos / DEFAULT_OPEN_STATE_DURATION_THRESHOLD * TIME_IN_OPEN_STATE_COEFFICIENT;

  return decimalFailureRating + decimalSlowCallRating + decimalSuccessCallRating + timeInOpenStateRating;
}
```

Реалізація


ThresholdCircuitBreaker
ThresholdCircuitBreakerConfiguration
ThresholdCircuitBreakerAspect
«create»
thresholdCircuitBreakerRegistry
ThresholdCircuitBreakerRegistry
allCircuitBreakers Set<CircuitBreaker>
«create»
ThresholdCircuitBreaker
tags Map<String, String>
circuitBreakerConfig CircuitBreakerConfig
name String
timestampUnit TimeUnit
name String
metrics Metrics
tags Map<String, String>
circuitBreakerConfig CircuitBreakerConfig
state State
currentTimestamp long
timestampUnit TimeUnit
eventPublisher EventPublisher


StateTransitionCalculator
calculateTransitionValue(SimpleMetrics, long) float
stateTransitionCalculator
«create»
ThresholdCircuitBreaker
transitionToForcedOpenState() void
onError(long, TimeUnit, Throwable) void
handlePossibleTransition(Either<Object, Throwable>) void
getTags() Map<String, String>
onResult(long, TimeUnit, Object?) void
transitionToOpenStateUntil(Instant) void
stateTransition(State, UnaryOperator<SimpleState>) void
getTimestampUnit() TimeUnit
getCurrentTimestamp() long
handleThrowable(long, TimeUnit, Throwable) void
tryAcquirePermission() boolean
transitionToClosedState() void
transitionToDisabledState() void
onSuccess(long, TimeUnit) void
transitionToOpenStateFor(Duration) void
acquirePermission() void
getState() State
releasePermission() void
getEventPublisher() EventPublisher
reset() void
transitionToOpenState() void
transitionToMetricsOnlyState() void
getCircuitBreakerConfig() CircuitBreakerConfig
getMetrics() Metrics
getName() String
transitionToHalfOpenState() void
«create»
SimpleState
tryAcquirePermission() boolean
acquirePermission() void
handlePossibleTransition(TransitionCheckResult) void
getMetrics() SimpleMetrics
onSuccess(long, TimeUnit) void
getState() State
releasePermission() void
onError(long, TimeUnit, Throwable) void
«create»
ClosedState
handlePossibleTransition(TransitionCheckResult) void
getState() State
tryAcquirePermission() boolean
toOpenState() void
onSuccess(long, TimeUnit) void
getMetrics() SimpleMetrics
onError(long, TimeUnit, Throwable) void
checkIfThresholdsExceeded(Result) void
acquirePermission() void
releasePermission() void
OpenState
getState() State
tryAcquirePermission() boolean
getMetrics() SimpleMetrics
handlePossibleTransition(TransitionCheckResult) void
releasePermission() void
onSuccess(long, TimeUnit) void
toClosedState() void
acquirePermission() void
onError(long, TimeUnit, Throwable) void
calculateTransitionRatingValue() float
circuitBreakerMetrics
circuitBreakerMetrics
SimpleMetrics
getNumberOfSuccessfulCalls() int
getSlowCallRate(Snapshot) float
getDecimalSlowCallRate() float
getDecimalSuccessRate() float
getNumberOfSlowCalls() int
onSuccess(long, TimeUnit) Result
checkIfThresholdsExceeded(Snapshot) Result
getNumberOfBufferedCalls() int
getNumberOfFailedCalls() int
onCallNotPermitted() void
getNumberOfSlowSuccessfulCalls() int
getFailureRate() float
checkNotExceedsMinimumNumberOfCalls(Snapshot) boolean
forClosed(CircuitBreakerConfig) SimpleMetrics
getSlowCallRate() float
getFailureRate(Snapshot) float
onError(long, TimeUnit) Result
getNumberOfSlowFailedCalls() int
getTargetCallNumberRate(Snapshot, int) float
getDecimalFailureRate() float
getNumberOfNotPermittedCalls() long

	Circuit Breaker		**(Success call rate)**
Мін. час успішної відповіді: 50 мс; Макс. час відповіді: 6 с; Мін. час перевірки стану за таймером: 1 с; Макс. час перевірки стану за таймером: 10 с; Ймовірність переходу у стан відмови: 0.2; Мін. час відновлення зі стану відмови: 1 с; Макс. час відновлення зі стану відмови: 10с.	Кількість запитів: 100 Розмір sliding window: 20 Час очікування у Open State (для традиційного Circuit Breaker): 3 с. Failure Rate поріг: 50% Slow Call поріг: 3 с. **Поріг рейтингу для переходу у стан Closed: 0.45**	Failure rate: 0.4 Slow call rate: 0.2 Success call rate: 0.3 Time in open state: 0.1	Традиційний CB: 63% Threshold CB: 74%
Мін. час успішної відповіді: 50 мс; Макс. час відповіді: 3 с; Мін. час перевірки стану за таймером: 1 с; Макс. час перевірки стану за таймером: 10 с; Ймовірність переходу у стан відмови: 0.3; Мін. час відновлення зі стану відмови: 1 с; Макс. час відновлення зі стану відмови: 10с.	Кількість запитів: 100 Розмір sliding window: 20 Час очікування у Open State (для традиційного Circuit Breaker): 5 с. Failure Rate Threshold: 40% Slow Call Threshold: 1.5 с. **Поріг рейтингу для переходу у стан Closed: 0.4**	Failure rate: 0.4 Slow call rate: 0.15 Success call rate: 0.35 Time in open state: 0.1	Традиційний CB: 51% Threshold CB: 61%

Поведінка віддаленого сервісу	**Конфігурація обох варіантів Circuit Breaker**	**Коефіцієнти**	**Результати (Success call rate)**
Мін. час успішної відповіді: 50 мс; Макс. час відповіді: 3 с; Мін. час перевірки стану за таймером: 1 с; Макс. час перевірки стану за таймером: 10 с; Ймовірність переходу у стан відмови: 0.3; Мін. час відновлення зі стану відмови: 1 с; Макс. час відновлення зі стану відмови: 10с.	Кількість запитів: 100 **Розмір sliding window: 10** Час очікування у Open State (для традиційного Circuit Breaker): 5 с. Failure Rate Threshold: 40% Slow Call Threshold: 1.5 с. **Поріг рейтингу для переходу у стан Closed: 0.4**	Failure rate: 0.4 Slow call rate: 0.15 Success call rate: 0.35 Time in open state: 0.1	Традиційний CB: 41% Threshold CB: 36%

Висновки

- Запропонована модель може використовуватись замість традиційного дизайну Circuit Breaker для досягнення більшої стабільності взаємодії між мікросервісами.
- Реалізація на основі фреймворку Resilience4j дає можливість “йти в ногу з часом” та використовувати інші сучасні підходи та патерни водночас із представленою імплементацією.
- Модель прогнозування станів Circuit Breaker дозволяє ще більше покладатись на результати взаємодій між сервісами в системі, в порівнянні з класичною реалізацією патерну.